Introduction to Randomized Algorithms

Subrahmanyam Kalyanasundaram
subruk@iith.ac.in

Department of Computer Science & Engineering
Indian Institute of Technology Hyderabad

Introduction to Graph and Geometric Algortihms
6 March 2014, IIT Roorkee

Outline

Outline

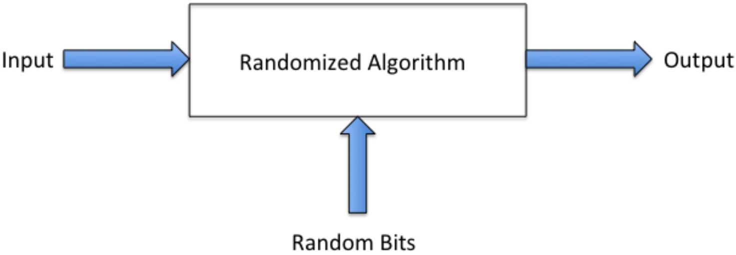
Deterministic Algorithms

- Goal: To solve a computational problem correctly and efficiently.
- Behaviour of the algorithm is determined completely by the input.
- Upon reruns, the algorithm executes in exactly the same manner.

Deterministic Algorithms

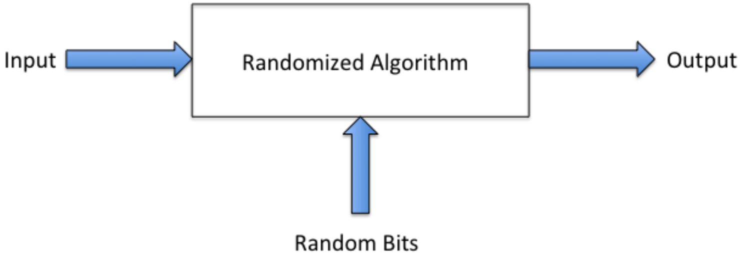
- Goal: To solve a computational problem correctly and efficiently.
- Behaviour of the algorithm is determined completely by the input.
- Upon reruns, the algorithm executes in exactly the same manner.

Randomized Algorithms



- In addition to the input, the algorithm execution depends on some random bits as well.
- Behaviour of the algorithm is not determined completely by the input.
- Upon reruns, the algorithm can execute in a different manner, even with the same input.

Randomized Algorithms



- In addition to the input, the algorithm execution depends on some random bits as well.
- Behaviour of the algorithm is not determined completely by the input.
- Upon reruns, the algorithm can execute in a different manner, even with the same input.

Intuition

- Deterministic algorithms can have certain “bad inputs”.
- Could make computation go to worst case running time.

- Most inputs aren’t so “bad”.
- Randomized algorithms use random bits to change the execution.
- Any given input is now unlikely to be bad.

- Another perspective: random bits choose one algorithm out of several ones.

Intuition

- Deterministic algorithms can have certain “bad inputs”.
- Could make computation go to worst case running time.

- Most inputs aren’t so “bad”.
- Randomized algorithms use random bits to change the execution.
- Any given input is now unlikely to be bad.

- Another perspective: random bits choose one algorithm out of several ones.

Intuition

- Deterministic algorithms can have certain “bad inputs”.
- Could make computation go to worst case running time.

- Most inputs aren’t so “bad”.
- Randomized algorithms use random bits to change the execution.
- Any given input is now unlikely to be bad.

- Another perspective: random bits choose one algorithm out of several ones.

Randomized vs. Deterministic

Deterministic algorithms

- Always correct answer.
- Always runs within the worst case running time.

Randomized Algorithms

- Gives the right answer.
- In good running time.
- Not necessarily always, but with good probability.

Randomized vs. Deterministic

Deterministic algorithms

- Always correct answer.
- Always runs within the worst case running time.

Randomized Algorithms

- Gives the right answer.
- In good running time.
- Not necessarily always, but with good probability.

Randomized vs. Deterministic

Deterministic algorithms

- Always correct answer.
- Always runs within the worst case running time.

Randomized Algorithms

- Gives the right answer.
- In good running time.
- Not necessarily always, but with good probability.

Broadly two types

Las Vegas Algorithms

- Correctness is guaranteed
- May not be fast always.
- Probability of worst case running time is small.
- Expected running time $<$ Worst case running time

Monte Carlo Algorithms

- Running time is fixed.
- Correctness of the algorithm need not be assured.
- Probability of an incorrect output is small.

Broadly two types

Las Vegas Algorithms

- Correctness is guaranteed
- May not be fast always.
- Probability of worst case running time is small.
- Expected running time $<$ Worst case running time

Monte Carlo Algorithms

- Running time is fixed.
- Correctness of the algorithm need not be assured.
- Probability of an incorrect output is small.

Broadly two types

Las Vegas Algorithms

- Correctness is guaranteed
- May not be fast always.
- Probability of worst case running time is small.
- Expected running time $<$ Worst case running time

Monte Carlo Algorithms

- Running time is fixed.
- Correctness of the algorithm need not be assured.
- Probability of an incorrect output is small.

Broadly two types

Las Vegas Algorithms

- Correctness is guaranteed
- May not be fast always.
- Probability of worst case running time is small.
- Expected running time $<$ Worst case running time

Monte Carlo Algorithms

- Running time is fixed.
- Correctness of the algorithm need not be assured.
- Probability of an incorrect output is small.

Advantages and Disadvantages

Advantages

- Simplicity
- Good performance
- In some cases, no deterministic algorithm exists.
- Adversary cannot choose a bad input.

Disadvantages

- Randomness is a resource.
- With some probability, we can get an incorrect output.
- With some probability, can perform in worst case time.

Advantages and Disadvantages

Advantages

- Simplicity
- Good performance
- In some cases, no deterministic algorithm exists.
- Adversary cannot choose a bad input.

Disadvantages

- Randomness is a resource.
- With some probability, we can get an incorrect output.
- With some probability, can perform in worst case time.

Advantages and Disadvantages

Advantages

- Simplicity
- Good performance
- In some cases, no deterministic algorithm exists.
- Adversary cannot choose a bad input.

Disadvantages

- Randomness is a resource.
- With some probability, we can get an incorrect output.
- With some probability, can perform in worst case time.

Probabilistic Analysis

- Algorithm output/performance can vary depending on random bits.
- Analysis will yield probabilistic statements.

- We need mathematical basis to analyze randomized algorithms.
- At times, the analysis could be long and complicated.
- The analysis could use mathematical tools of varying difficulty.
- But most randomized algorithms are extremely simple to describe and program.

Probabilistic Analysis

- Algorithm output/performance can vary depending on random bits.
- Analysis will yield probabilistic statements.

- We need mathematical basis to analyze randomized algorithms.
- At times, the analysis could be long and complicated.
- The analysis could use mathematical tools of varying difficulty.
- But most randomized algorithms are extremely simple to describe and program.

Preliminaries

- Probability is over the distribution of the random bits.
- Probability is not over the input distribution.
- For a random variable X, $\Pr(X = x)$ denotes the probability with which X takes the value x.
- $E(X)$ denotes the expectation of the random variable X.

Preliminaries

- Probability is over the distribution of the random bits.
- Probability is not over the input distribution.

- For a random variable X, $\Pr(X = x)$ denotes the probability with which X takes the value x.

- $E(X)$ denotes the expectation of the random variable X.

Preliminaries

- Probability is over the distribution of the random bits.
- Probability is not over the input distribution.
- For a random variable X, $\Pr(X = x)$ denotes the probability with which X takes the value x.
- $E(X)$ denotes the expectation of the random variable X.

Outline

Is a given polynomial equal to zero?

Polynomial Identity Testing

Is a given polynomial $P(x)$ identically equal to 0?

Another form

Are two given polynomials $F(x)$ and $G(x)$ identically equal to one another?

$$F(x) \stackrel{?}{\equiv} G(x)$$

Is a given polynomial equal to zero?

Polynomial Identity Testing

Is a given polynomial $P(x)$ identically equal to 0?

Another form

Are two given polynomials $F(x)$ and $G(x)$ identically equal to one another?

$$F(x) \stackrel{?}{\equiv} G(x)$$

Polynomial Identity Testing

What can we do deterministically?

- Want to check if
  $F(x) = (x-1)(x+3)(x-6) \equiv x^3 + 4x^2 - 12x + 18 = G(x)$.

Deterministic Algorithm

- Convert the two polynomials to a standard format.
- Check if they are the same.

- If polynomials are of degree d, this requires $\Theta(d^2)$ time.
- Always correct.

- Want to check if
  $F(x) = (x-1)(x+3)(x-6) \equiv x^3 + 4x^2 - 12x + 18 = G(x)$.

Deterministic Algorithm

- Convert the two polynomials to a standard format.
- Check if they are the same.

- If polynomials are of degree d, this requires $\Theta(d^2)$ time.
- Always correct.

Polynomial Identity Testing

What can we do deterministically?

- Want to check if $F(x) = (x-1)(x+3)(x-6) \equiv x^3 + 4x^2 - 12x + 18 = G(x)$.

Deterministic Algorithm

- Convert the two polynomials to a standard format.
- Check if they are the same.

- If polynomials are of degree d, this requires $\Theta(d^2)$ time.
- Always correct.

Polynomial Identity Testing

What can we do deterministically?

- Want to check if
  $F(x) = (x-1)(x+3)(x-6) \equiv x^3 + 4x^2 - 12x + 18 = G(x)$.

Deterministic Algorithm

- Convert the two polynomials to a standard format.
- Check if they are the same.

- If polynomials are of degree d, this requires $\Theta(d^2)$ time.
- Always correct.

- Want to check if
  $F(x) = (x-1)(x+3)(x-6) \equiv x^3 + 4x^2 - 12x + 18 = G(x)$.

Deterministic Algorithm

- Convert the two polynomials to a standard format.
- Check if they are the same.

- If polynomials are of degree d, this requires $\Theta(d^2)$ time.
- Always correct.

Polynomial Identity Testing

A Randomized Algorithm

- Choose a value r from a set S of $100d$ possible values.
- Evaluate $F(r)$ and $G(r)$.
- Check if $F(r) = G(r)$.

- Running time: How long does the evaluation take?
- Can use Horner's Method to evaluate $F(r)$ and $G(r)$ in $\Theta(d)$ time.

- What about correctness?
- If $F(x) \equiv G(x)$, then $F(r) = G(r)$ for any r.
- What if $F(x) \not\equiv G(x)$?

Polynomial Identity Testing

A Randomized Algorithm

- Choose a value r from a set S of $100d$ possible values.
- Evaluate $F(r)$ and $G(r)$.
- Check if $F(r) = G(r)$.

- Running time: How long does the evaluation take?
- Can use Horner's Method to evaluate $F(r)$ and $G(r)$ in $\Theta(d)$ time.

$$6x^3 + 12x^2 - 10x + 23 = ((6x + 12)x - 10)x + 23$$

- What about correctness?
- If $F(x) \equiv G(x)$, then $F(r) = G(r)$ for any r.
- What if $F(x) \not\equiv G(x)$?

Polynomial Identity Testing

A Randomized Algorithm

- Choose a value r from a set S of $100d$ possible values.
- Evaluate $F(r)$ and $G(r)$.
- Check if $F(r) = G(r)$.

- Running time: How long does the evaluation take?
- Can use Horner's Method to evaluate $F(r)$ and $G(r)$ in $\Theta(d)$ time.

- What about correctness?
- If $F(x) \equiv G(x)$, then $F(r) = G(r)$ for any r.
- What if $F(x) \not\equiv G(x)$?

Polynomial Identity Testing

A Randomized Algorithm

- Choose a value r from a set S of $100d$ possible values.
- Evaluate $F(r)$ and $G(r)$.
- Check if $F(r) = G(r)$.

- Running time: How long does the evaluation take?
- Can use Horner's Method to evaluate $F(r)$ and $G(r)$ in $\Theta(d)$ time.

- What about correctness?
- If $F(x) \equiv G(x)$, then $F(r) = G(r)$ for any r.
- What if $F(x) \not\equiv G(x)$?

Polynomial Identity Testing

A Randomized Algorithm

- Choose a value r from a set S of $100d$ possible values.
- Evaluate $F(r)$ and $G(r)$.
- Check if $F(r) = G(r)$.

- Running time: How long does the evaluation take?
- Can use Horner's Method to evaluate $F(r)$ and $G(r)$ in $\Theta(d)$ time.

- What about correctness?
- If $F(x) \equiv G(x)$, then $F(r) = G(r)$ for any r.
- What if $F(x) \not\equiv G(x)$?

Analysis of Correctness

Fundamental Theorem of Algebra

If $P(x)$ is a polynomial of degree d, it has at most d roots.

- Let $F(x) \not\equiv G(x)$.
- By the above theorem, $F(r) - G(r) = 0$ for $\leq d$ values of $r \in S$.
- There are at most d values of r which can lead to a wrong answer.
- If $|S| \geq 100d$, then $\Pr(F(r) = G(r)) \leq 1/100$.
- Probability of error is $\leq 1/100$.

- Not happy with the probability of success? Then repeat!

Analysis of Correctness

Fundamental Theorem of Algebra

If $P(x)$ is a polynomial of degree d, it has at most d roots.

- Let $F(x) \not\equiv G(x)$.
- By the above theorem, $F(r) - G(r) = 0$ for $\leq d$ values of $r \in S$.
- There are at most d values of r which can lead to a wrong answer.
- If $|S| \geq 100d$, then $\Pr(F(r) = G(r)) \leq 1/100$.
- Probability of error is $\leq 1/100$.

- Not happy with the probability of success? Then repeat!

Analysis of Correctness

Fundamental Theorem of Algebra

If $P(x)$ is a polynomial of degree d, it has at most d roots.

- Let $F(x) \not\equiv G(x)$.
- By the above theorem, $F(r) - G(r) = 0$ for $\leq d$ values of $r \in S$.
- There are at most d values of r which can lead to a wrong answer.
- If $|S| \geq 100d$, then $\Pr(F(r) = G(r)) \leq 1/100$.
- Probability of error is $\leq 1/100$.

- Not happy with the probability of success? Then repeat!

Analysis of Correctness

Fundamental Theorem of Algebra

If $P(x)$ is a polynomial of degree d, it has at most d roots.

- Let $F(x) \not\equiv G(x)$.
- By the above theorem, $F(r) - G(r) = 0$ for $\leq d$ values of $r \in S$.
- There are at most d values of r which can lead to a wrong answer.
- If $|S| \geq 100d$, then $\Pr(F(r) = G(r)) \leq 1/100$.
- Probability of error is $\leq 1/100$.
- Not happy with the probability of success? Then repeat!

Analysis of Correctness

Fundamental Theorem of Algebra

If $P(x)$ is a polynomial of degree d, it has at most d roots.

- Let $F(x) \not\equiv G(x)$.
- By the above theorem, $F(r) - G(r) = 0$ for $\leq d$ values of $r \in S$.
- There are at most d values of r which can lead to a wrong answer.
- If $|S| \geq 100d$, then $\Pr(F(r) = G(r)) \leq 1/100$.
- Probability of error is $\leq 1/100$.

- Not happy with the probability of success? Then repeat!

Analysis of Correctness

Fundamental Theorem of Algebra

If $P(x)$ is a polynomial of degree d, it has at most d roots.

- Let $F(x) \not\equiv G(x)$.
- By the above theorem, $F(r) - G(r) = 0$ for $\leq d$ values of $r \in S$.
- There are at most d values of r which can lead to a wrong answer.
- If $|S| \geq 100d$, then $\Pr(F(r) = G(r)) \leq 1/100$.
- Probability of error is $\leq 1/100$.

- Not happy with the probability of success? Then repeat!

Boosting the Probability of Success

- Saw a Monte Carlo algorithm with Pr(success) $\geq 1 - 1/100$.

Boosting

Repeating a Monte Carlo algorithm to achieve a better probability of success.

Boosted Algorithm

- Choose two values r_1, r_2 from a set S of $100d$ possible values.
- Evaluate $F(r_1), F(r_2), G(r_1)$ and $G(r_2)$.
- Check if $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.
- Report "same" if both $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.

Boosting the Probability of Success

- Saw a Monte Carlo algorithm with $\Pr(\text{success}) \geq 1 - 1/100$.

Boosting

Repeating a Monte Carlo algorithm to achieve a better probability of success.

Boosted Algorithm

- Choose two values r_1, r_2 from a set S of $100d$ possible values.
- Evaluate $F(r_1), F(r_2), G(r_1)$ and $G(r_2)$.
- Check if $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.
- Report "same" if both $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.

Boosting the Probability of Success

- Saw a Monte Carlo algorithm with $\Pr(\text{success}) \geq 1 - 1/100$.

Boosting

Repeating a Monte Carlo algorithm to achieve a better probability of success.

Boosted Algorithm

- Choose two values r_1, r_2 from a set S of $100d$ possible values.
- Evaluate $F(r_1), F(r_2), G(r_1)$ and $G(r_2)$.
- Check if $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.
- Report "same" if both $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.

Boosting the Probability of Success

Boosted Algorithm

- Choose two values r_1, r_2 from a set S of $100d$ possible values.
- Evaluate $F(r_1), F(r_2), G(r_1)$ and $G(r_2)$.
- Check if $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.
- Report "same" if both $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.

- Suppose $F(x) \not\equiv G(x)$.
- If we run two independent trials

$$\Pr(F(r) = G(r) \text{ in both trials}) \leq 1/100^2$$

- Boosting is a standard technique for achieving desired probability with Monte Carlo algorithms.

Boosting the Probability of Success

Boosted Algorithm

- Choose two values r_1, r_2 from a set S of $100d$ possible values.
- Evaluate $F(r_1), F(r_2), G(r_1)$ and $G(r_2)$.
- Check if $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.
- Report "same" if both $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.

- Suppose $F(x) \not\equiv G(x)$.
- If we run two independent trials

$$\Pr(F(r) = G(r) \text{ in both trials}) \leq 1/100^2$$

- Boosting is a standard technique for achieving desired probability with Monte Carlo algorithms.

Boosting the Probability of Success

Boosted Algorithm

- Choose two values r_1, r_2 from a set S of $100d$ possible values.
- Evaluate $F(r_1), F(r_2), G(r_1)$ and $G(r_2)$.
- Check if $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.
- Report "same" if both $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.

- Suppose $F(x) \not\equiv G(x)$.
- If we run two independent trials

$$\Pr(F(r) = G(r) \text{ in both trials}) \leq 1/100^2$$

- Boosting is a standard technique for achieving desired probability with Monte Carlo algorithms.

Boosting the Probability of Success

Boosted Algorithm

- Choose two values r_1, r_2 from a set S of $100d$ possible values.
- Evaluate $F(r_1), F(r_2), G(r_1)$ and $G(r_2)$.
- Check if $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.
- Report "same" if both $F(r_1) = G(r_1)$ and $F(r_2) = G(r_2)$.

- Suppose $F(x) \not\equiv G(x)$.
- If we run two independent trials

$$\Pr(F(r) = G(r) \text{ in both trials}) \leq 1/100^2$$

- Boosting is a standard technique for achieving desired probability with Monte Carlo algorithms.

Multivariate Case

Polynomial Identity Testing: Multivariate Case

Is a given polynomial $P(x_1, x_2, \ldots, x_n)$ identically equal to 0?

- No known deterministic polynomial time algorithm for the multivariate case.
- Multiplying out a polynomial can result in exponentially many terms.
- For the multivariate case, we need a stronger theorem than the Fundamental Theorem of Algebra.

Multivariate Case

Polynomial Identity Testing: Multivariate Case

Is a given polynomial $P(x_1, x_2, \ldots, x_n)$ identically equal to 0?

- No known deterministic polynomial time algorithm for the multivariate case.
- Multiplying out a polynomial can result in exponentially many terms.
- For the multivariate case, we need a stronger theorem than the Fundamental Theorem of Algebra.

Multivariate Case

Polynomial Identity Testing: Multivariate Case

Is a given polynomial $P(x_1, x_2, \ldots, x_n)$ identically equal to 0?

- No known deterministic polynomial time algorithm for the multivariate case.
- Multiplying out a polynomial can result in exponentially many terms.
- For the multivariate case, we need a stronger theorem than the Fundamental Theorem of Algebra.

Multivariate Case

Polynomial Identity Testing: Multivariate Case

Is a given polynomial $P(x_1, x_2, \ldots, x_n)$ identically equal to 0?

- No known deterministic polynomial time algorithm for the multivariate case.
- Multiplying out a polynomial can result in exponentially many terms.
- For the multivariate case, we need a stronger theorem than the Fundamental Theorem of Algebra.

Multivariate Case

A Randomized Algorithm

- Choose $(r_1, r_2, \ldots, r_n)$ from a set S of $100d$ possible values.
- Evaluate $P(r_1, r_2, \ldots, r_n)$.
- Report that $P(x_1, x_2, \ldots, x_n) \equiv 0$ if $P(r_1, r_2, \ldots, r_n) = 0$.

DeMillo-Lipton-Schwartz-Zippel Lemma

In the above setting, $\Pr(P(r_1, r_2, \ldots, r_n) = 0) \leq d/|S|$

- Probability of failure is $\leq 1/100$.
- Assumption: $P(r_1, r_2, \ldots, r_n)$ can be efficiently evaluated.
- Randomization, indeed, seems to help.

Multivariate Case

A Randomized Algorithm

- Choose $(r_1, r_2, \ldots, r_n)$ from a set S of $100d$ possible values.
- Evaluate $P(r_1, r_2, \ldots, r_n)$.
- Report that $P(x_1, x_2, \ldots, x_n) \equiv 0$ if $P(r_1, r_2, \ldots, r_n) = 0$.

DeMillo-Lipton-Schwartz-Zippel Lemma

In the above setting, $\Pr(P(r_1, r_2, \ldots, r_n) = 0) \leq d/|S|$

- Probability of failure is $\leq 1/100$.
- Assumption: $P(r_1, r_2, \ldots, r_n)$ can be efficiently evaluated.
- Randomization, indeed, seems to help.

Multivariate Case

A Randomized Algorithm

- Choose $(r_1, r_2, \ldots, r_n)$ from a set S of $100d$ possible values.
- Evaluate $P(r_1, r_2, \ldots, r_n)$.
- Report that $P(x_1, x_2, \ldots, x_n) \equiv 0$ if $P(r_1, r_2, \ldots, r_n) = 0$.

DeMillo-Lipton-Schwartz-Zippel Lemma

In the above setting, $\Pr(P(r_1, r_2, \ldots, r_n) = 0) \leq d/|S|$

- Probability of failure is $\leq 1/100$.
- Assumption: $P(r_1, r_2, \ldots, r_n)$ can be efficiently evaluated.
- Randomization, indeed, seems to help.

Multivariate Case

A Randomized Algorithm

- Choose $(r_1, r_2, \ldots, r_n)$ from a set S of $100d$ possible values.
- Evaluate $P(r_1, r_2, \ldots, r_n)$.
- Report that $P(x_1, x_2, \ldots, x_n) \equiv 0$ if $P(r_1, r_2, \ldots, r_n) = 0$.

DeMillo-Lipton-Schwartz-Zippel Lemma

In the above setting, $\Pr(P(r_1, r_2, \ldots, r_n) = 0) \leq d/|S|$

- Probability of failure is $\leq 1/100$.
- Assumption: $P(r_1, r_2, \ldots, r_n)$ can be efficiently evaluated.
- Randomization, indeed, seems to help.

Multivariate Case

A Randomized Algorithm

- Choose $(r_1, r_2, \ldots, r_n)$ from a set S of $100d$ possible values.
- Evaluate $P(r_1, r_2, \ldots, r_n)$.
- Report that $P(x_1, x_2, \ldots, x_n) \equiv 0$ if $P(r_1, r_2, \ldots, r_n) = 0$.

DeMillo-Lipton-Schwartz-Zippel Lemma

In the above setting, $\Pr(P(r_1, r_2, \ldots, r_n) = 0) \leq d/|S|$

- Probability of failure is $\leq 1/100$.
- Assumption: $P(r_1, r_2, \ldots, r_n)$ can be efficiently evaluated.
- Randomization, indeed, seems to help.

Outline

Quicksort

Problem

Given an array A of n elements, arrange the elements in increasing order.

Quicksort(A, s, t)

1. If $s \geq t$, exit.
2. Choose pivot p from $\{s, s+1, \ldots, t\}$
3. $q =$Partition(A, s, t, p). Partition(A, s, t, p) partitions $A(s, t)$ in place into less than pivot, pivot and greater than pivot. It also returns the correct index of p.
4. Quicksort$(A, s, q-1)$
5. Quicksort$(A, q+1, t)$

Quicksort

Problem

Given an array A of n elements, arrange the elements in increasing order.

Quicksort(A, s, t)

1. If $s \geq t$, exit.
2. Choose pivot p from $\{s, s+1, \ldots, t\}$
3. $q =$Partition(A, s, t, p). Partition(A, s, t, p) partitions $A(s, t)$ in place into less than pivot, pivot and greater than pivot. It also returns the correct index of p.
4. Quicksort($A, s, q-1$)
5. Quicksort($A, q+1, t$)

Quicksort

Problem

Given an array A of n elements, arrange the elements in increasing order.

Quicksort(A, s, t)

1. If $s \geq t$, exit.
2. Choose pivot p from $\{s, s+1, \ldots, t\}$
3. $q =$Partition(A, s, t, p). Partition(A, s, t, p) partitions $A(s, t)$ in place into less than pivot, pivot and greater than pivot. It also returns the correct index of p.
4. Quicksort($A, s, q-1$)
5. Quicksort($A, q+1, t$)

Quicksort

Problem

Given an array A of n elements, arrange the elements in increasing order.

Quicksort(A, s, t)

1. If $s \geq t$, exit.
2. Choose pivot p from $\{s, s+1, \ldots, t\}$
3. $q =$Partition(A, s, t, p). Partition(A, s, t, p) partitions $A(s, t)$ in place into less than pivot, pivot and greater than pivot. It also returns the correct index of p.
4. Quicksort($A, s, q-1$)
5. Quicksort($A, q+1, t$)

Quicksort

Problem

Given an array A of n elements, arrange the elements in increasing order.

Quicksort(A, s, t)

1. If $s \geq t$, exit.
2. Choose pivot p from $\{s, s+1, \ldots, t\}$
3. $q =$Partition(A, s, t, p). Partition(A, s, t, p) partitions $A(s, t)$ in place into less than pivot, pivot and greater than pivot. It also returns the correct index of p.
4. Quicksort($A, s, q-1$)
5. Quicksort($A, q+1, t$)

Quicksort

Problem

Given an array A of n elements, arrange the elements in increasing order.

Quicksort(A, s, t)

1. If $s \geq t$, exit.
2. Choose pivot p from $\{s, s+1, \ldots, t\}$
3. $q =$Partition(A, s, t, p). Partition(A, s, t, p) partitions $A(s, t)$ in place into less than pivot, pivot and greater than pivot. It also returns the correct index of p.
4. Quicksort($A, s, q-1$)
5. Quicksort($A, q+1, t$)

Quicksort

Problem

Given an array A of n elements, arrange the elements in increasing order.

Quicksort(A, s, t)

1. If $s \geq t$, exit.
2. Choose pivot p from $\{s, s+1, \ldots, t\}$
3. $q =$Partition(A, s, t, p). Partition(A, s, t, p) partitions $A(s, t)$ in place into less than pivot, pivot and greater than pivot. It also returns the correct index of p.
4. Quicksort$(A, s, q-1)$
5. Quicksort$(A, q+1, t)$

Quicksort

Problem

Given an array A of n elements, arrange the elements in increasing order.

Quicksort(A, s, t)

1. If $s \geq t$, exit.
2. Choose pivot p from $\{s, s+1, \ldots, t\}$
3. $q =$Partition(A, s, t, p). Partition(A, s, t, p) partitions $A(s, t)$ in place into less than pivot, pivot and greater than pivot. It also returns the correct index of p.
4. Quicksort($A, s, q-1$)
5. Quicksort($A, q+1, t$)

Deterministic Quicksort

Quicksort(A, s, t)

- If $s \geq t$, exit.
- Deterministically choose pivot p from $\{s, s+1, \ldots, t\}$
- $q =$Partition(A, s, t, p).
- Quicksort($A, s, q-1$)
- Quicksort($A, q+1, t$)

- For instance, pivot p is always the first element.
- The running time is determined by the number of comparisons.
- Any deterministic pivot rule requires worst case $\Omega(n^2)$ comparisons.
- One can come up with a bad input order for any deterministic pivot rule.
- Can randomization help?

Deterministic Quicksort

Quicksort(A, s, t)

- If $s \geq t$, exit.
- Deterministically choose pivot p from $\{s, s+1, \ldots, t\}$
- $q =$Partition(A, s, t, p).
- Quicksort($A, s, q-1$)
- Quicksort($A, q+1, t$)

- For instance, pivot p is always the first element.
- The running time is determined by the number of comparisons.
- Any deterministic pivot rule requires worst case $\Omega(n^2)$ comparisons.
- One can come up with a bad input order for any deterministic pivot rule.
- Can randomization help?

Deterministic Quicksort

Quicksort(A, s, t)

- If $s \geq t$, exit.
- Deterministically choose pivot p from $\{s, s+1, \ldots, t\}$
- $q =$Partition(A, s, t, p).
- Quicksort($A, s, q-1$)
- Quicksort($A, q+1, t$)

- For instance, pivot p is always the first element.
- The running time is determined by the number of comparisons.
- Any deterministic pivot rule requires worst case $\Omega(n^2)$ comparisons.
- One can come up with a bad input order for any deterministic pivot rule.
- Can randomization help?

Deterministic Quicksort

Quicksort(A, s, t)

- If $s \geq t$, exit.
- Deterministically choose pivot p from $\{s, s+1, \ldots, t\}$
- $q =$Partition(A, s, t, p).
- Quicksort($A, s, q-1$)
- Quicksort($A, q+1, t$)

- For instance, pivot p is always the first element.
- The running time is determined by the number of comparisons.
- Any deterministic pivot rule requires worst case $\Omega(n^2)$ comparisons.
- One can come up with a bad input order for any deterministic pivot rule.
- Can randomization help?

Deterministic Quicksort

Quicksort(A, s, t)

- If $s \geq t$, exit.
- Deterministically choose pivot p from $\{s, s+1, \ldots, t\}$
- $q =$Partition(A, s, t, p).
- Quicksort($A, s, q-1$)
- Quicksort($A, q+1, t$)

- For instance, pivot p is always the first element.
- The running time is determined by the number of comparisons.
- Any deterministic pivot rule requires worst case $\Omega(n^2)$ comparisons.
- One can come up with a bad input order for any deterministic pivot rule.
- Can randomization help?

Why randomize?

- Worst case occurs when we repeatedly choose the smallest/largest number as pivot.
- A good pivot separates the array into two (roughly) equal parts.
- If pivot gives a $[n/10, 9n/10]$-split, we get the recurrence.

$$T(n) = T(n/10) + T(9n/10) + cn$$

- Even this gives us $\Theta(n \log n)$ number of comparisons.
- A random pivot is likely to work with probability 0.8.
- This is still an intuition.

Why randomize?

- Worst case occurs when we repeatedly choose the smallest/largest number as pivot.
- A good pivot separates the array into two (roughly) equal parts.
- If pivot gives a $[n/10, 9n/10]$-split, we get the recurrence.

$$T(n) = T(n/10) + T(9n/10) + cn$$

- Even this gives us $\Theta(n \log n)$ number of comparisons.
- A random pivot is likely to work with probability 0.8.
- This is still an intuition.

Why randomize?

- Worst case occurs when we repeatedly choose the smallest/largest number as pivot.
- A good pivot separates the array into two (roughly) equal parts.
- If we choose the median as the pivot, the recurrence for number of comparisons is
$$T(n) = 2T(n/2) + cn$$
- This solves to $\Theta(n \log n)$
- If pivot gives a $[n/10, 9n/10]$-split, we get the recurrence.
$$T(n) = T(n/10) + T(9n/10) + cn$$
- Even this gives us $\Theta(n \log n)$ number of comparisons.
- A random pivot is likely to work with probability 0.8.
- This is still an intuition.

Why randomize?

- Worst case occurs when we repeatedly choose the smallest/largest number as pivot.
- A good pivot separates the array into two (roughly) equal parts.
- If we choose the median as the pivot, the recurrence for number of comparisons is
$$T(n) = 2T(n/2) + cn$$
- This solves to $\Theta(n \log n)$
- If pivot gives a $[n/10, 9n/10]$-split, we get the recurrence.
$$T(n) = T(n/10) + T(9n/10) + cn$$
- Even this gives us $\Theta(n \log n)$ number of comparisons.
- A random pivot is likely to work with probability 0.8.
- This is still an intuition.

Why randomize?

- Worst case occurs when we repeatedly choose the smallest/largest number as pivot.
- A good pivot separates the array into two (roughly) equal parts.
- If pivot gives a $[n/10, 9n/10]$-split, we get the recurrence.

$$T(n) = T(n/10) + T(9n/10) + cn$$

- Even this gives us $\Theta(n \log n)$ number of comparisons.
- A random pivot is likely to work with probability 0.8.
- This is still an intuition.

Why randomize?

- Worst case occurs when we repeatedly choose the smallest/largest number as pivot.
- A good pivot separates the array into two (roughly) equal parts.
- If pivot gives a $[n/10, 9n/10]$-split, we get the recurrence.

$$T(n) = T(n/10) + T(9n/10) + cn$$

- Even this gives us $\Theta(n \log n)$ number of comparisons.

- A random pivot is likely to work with probability 0.8.
- This is still an intuition.

Why randomize?

- Worst case occurs when we repeatedly choose the smallest/largest number as pivot.
- A good pivot separates the array into two (roughly) equal parts.
- If pivot gives a $[n/10, 9n/10]$-split, we get the recurrence.

$$T(n) = T(n/10) + T(9n/10) + cn$$

- Even this gives us $\Theta(n \log n)$ number of comparisons.

- A random pivot is likely to work with probability 0.8.
- This is still an intuition.

Why randomize?

- Worst case occurs when we repeatedly choose the smallest/largest number as pivot.
- A good pivot separates the array into two (roughly) equal parts.
- If pivot gives a $[n/10, 9n/10]$-split, we get the recurrence.

$$T(n) = T(n/10) + T(9n/10) + cn$$

- Even this gives us $\Theta(n \log n)$ number of comparisons.

- A random pivot is likely to work with probability 0.8.
- This is still an intuition.

Randomized Quicksort

Quicksort(A, s, t)

- If $s \geq t$, exit.
- Choose pivot p uniformly at random from $\{s, s+1, \ldots, t\}$
- $q =$Partition(A, s, t, p).
- Quicksort($A, s, q-1$)
- Quicksort($A, q+1, t$)

Analysis of Randomized Quicksort

- Let the numbers in A be $z_1 < z_2 < \ldots < z_n$.

- Let $X_{i,j}$ denote an indicator random variable for all $1 \leq i < j \leq n$.
- If z_i is compared to z_j during the execution of the algorithm, $X_{i,j} = 1$.
- Otherwise $X_{i,j} = 0$

The total no. of comparisons X is given by

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j}$$

- Correct because $X_{i,j}$ takes only values from $\{0, 1\}$.
- Also because no two z_i and z_j are compared more than once.

Analysis of Randomized Quicksort

- Let the numbers in A be $z_1 < z_2 < \ldots < z_n$.

- Let $X_{i,j}$ denote an indicator random variable for all $1 \leq i < j \leq n$.
- If z_i is compared to z_j during the execution of the algorithm, $X_{i,j} = 1$.
- Otherwise $X_{i,j} = 0$

The total no. of comparisons X is given by

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j}$$

- Correct because $X_{i,j}$ takes only values from $\{0, 1\}$.
- Also because no two z_i and z_j are compared more than once.

Analysis of Randomized Quicksort

- Let the numbers in A be $z_1 < z_2 < \ldots < z_n$.

- Let $X_{i,j}$ denote an indicator random variable for all $1 \leq i < j \leq n$.
- If z_i is compared to z_j during the execution of the algorithm, $X_{i,j} = 1$.
- Otherwise $X_{i,j} = 0$

The total no. of comparisons X is given by

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j}$$

- Correct because $X_{i,j}$ takes only values from $\{0, 1\}$.
- Also because no two z_i and z_j are compared more than once.

Analysis of Randomized Quicksort

- Let the numbers in A be $z_1 < z_2 < \ldots < z_n$.

- Let $X_{i,j}$ denote an indicator random variable for all $1 \leq i < j \leq n$.
- If z_i is compared to z_j during the execution of the algorithm, $X_{i,j} = 1$.
- Otherwise $X_{i,j} = 0$

The total no. of comparisons X is given by

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j}$$

- Correct because $X_{i,j}$ takes only values from $\{0, 1\}$.
- Also because no two z_i and z_j are compared more than once.

Analysis of Randomized Quicksort

- Need to calculate expected number of comparisons $E(X)$.

Linearity of Expectations

$$E(X) = E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{i,j}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E(X_{i,j})$$

- For indicator random variable, $E(X_{i,j}) = \Pr(X_{i,j} = 1)$
- What is the probability that z_i was compared to z_j?

Analysis of Randomized Quicksort

- Need to calculate expected number of comparisons $E(X)$.

Linearity of Expectations

$$E(X) = E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{i,j}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E(X_{i,j})$$

- For indicator random variable, $E(X_{i,j}) = \Pr(X_{i,j} = 1)$
- What is the probability that z_i was compared to z_j?

Analysis of Randomized Quicksort

- Need to calculate expected number of comparisons $E(X)$.

Linearity of Expectations

$$E(X) = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{i,j}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(X_{i,j})$$

- For indicator random variable, $E(X_{i,j}) = \Pr(X_{i,j} = 1)$
- What is the probability that z_i was compared to z_j?

Analysis of Randomized Quicksort

- Need to calculate expected number of comparisons $E(X)$.

Linearity of Expectations

$$E(X) = E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{i,j}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E(X_{i,j})$$

- For indicator random variable, $E(X_{i,j}) = \Pr(X_{i,j} = 1)$
- What is the probability that z_i was compared to z_j?

Analysis of Randomized Quicksort

- Let $Z_{i,j} = \{z_i, z_{i+1}, \ldots, z_j\}$
- z_i is compared to z_j if and only if one of them is chosen as pivot.

Claim

$X_{i,j} = 1$ (z_i is compared to z_j) if and only if the first pivot chosen from $Z_{i,j}$ is z_i or z_j.

- As long as pivots in $Z_{i,j}$ are not chosen, z_i and z_j are never separated by the algorithm.
- If z_i or z_j is the first pivot chosen from $Z_{i,j}$, then z_i is compared to z_j.
- If the first pivot is from $Z_{i,j} \setminus \{z_i, z_j\}$, then z_i and z_j are never compared.

Analysis of Randomized Quicksort

- Let $Z_{i,j} = \{z_i, z_{i+1}, \ldots, z_j\}$
- z_i is compared to z_j if and only if one of them is chosen as pivot.

Claim

$X_{i,j} = 1$ (z_i is compared to z_j) if and only if the first pivot chosen from $Z_{i,j}$ is z_i or z_j.

- As long as pivots in $Z_{i,j}$ are not chosen, z_i and z_j are never separated by the algorithm.
- If z_i or z_j is the first pivot chosen from $Z_{i,j}$, then z_i is compared to z_j.
- If the first pivot is from $Z_{i,j} \backslash \{z_i, z_j\}$, then z_i and z_j are never compared.

Analysis of Randomized Quicksort

- Let $Z_{i,j} = \{z_i, z_{i+1}, \ldots, z_j\}$
- z_i is compared to z_j if and only if one of them is chosen as pivot.

Claim

$X_{i,j} = 1$ (z_i is compared to z_j) if and only if the first pivot chosen from $Z_{i,j}$ is z_i or z_j.

- As long as pivots in $Z_{i,j}$ are not chosen, z_i and z_j are never separated by the algorithm.
- If z_i or z_j is the first pivot chosen from $Z_{i,j}$, then z_i is compared to z_j.
- If the first pivot is from $Z_{i,j} \backslash \{z_i, z_j\}$, then z_i and z_j are never compared.

Analysis of Randomized Quicksort

- Let $Z_{i,j} = \{z_i, z_{i+1}, \ldots, z_j\}$
- z_i is compared to z_j if and only if one of them is chosen as pivot.

Claim

$X_{i,j} = 1$ (z_i is compared to z_j) if and only if the first pivot chosen from $Z_{i,j}$ is z_i or z_j.

- As long as pivots in $Z_{i,j}$ are not chosen, z_i and z_j are never separated by the algorithm.
- If z_i or z_j is the first pivot chosen from $Z_{i,j}$, then z_i is compared to z_j.
- If the first pivot is from $Z_{i,j} \setminus \{z_i, z_j\}$, then z_i and z_j are never compared.

Analysis of Randomized Quicksort

- Let $Z_{i,j} = \{z_i, z_{i+1}, \ldots, z_j\}$
- z_i is compared to z_j if and only if one of them is chosen as pivot.

Claim

$X_{i,j} = 1$ (z_i is compared to z_j) if and only if the first pivot chosen from $Z_{i,j}$ is z_i or z_j.

- As long as pivots in $Z_{i,j}$ are not chosen, z_i and z_j are never separated by the algorithm.
- If z_i or z_j is the first pivot chosen from $Z_{i,j}$, then z_i is compared to z_j.
- If the first pivot is from $Z_{i,j} \setminus \{z_i, z_j\}$, then z_i and z_j are never compared.

Analysis of Randomized Quicksort

- Let $Z_{i,j} = \{z_i, z_{i+1}, \ldots, z_j\}$
- z_i is compared to z_j if and only if one of them is chosen as pivot.

Claim

$X_{i,j} = 1$ (z_i is compared to z_j) if and only if the first pivot chosen from $Z_{i,j}$ is z_i or z_j.

- As long as pivots in $Z_{i,j}$ are not chosen, z_i and z_j are never separated by the algorithm.
- If z_i or z_j is the first pivot chosen from $Z_{i,j}$, then z_i is compared to z_j.
- If the first pivot is from $Z_{i,j} \setminus \{z_i, z_j\}$, then z_i and z_j are never compared.

Analysis of Randomized Quicksort

- What is the probability that z_i was compared to z_j?
- What is the probability that z_i or z_j is the first chosen pivot from $Z_{i,j}$?
- Since $|Z_{i,j}| = j - i + 1$,

$$E(X_{i,j}) = \Pr(X_{i,j} = 1) = 2/(j - i + 1).$$

Analysis of Randomized Quicksort

- What is the probability that z_i was compared to z_j?
- What is the probability that z_i or z_j is the first chosen pivot from $Z_{i,j}$?
- Since $|Z_{i,j}| = j - i + 1$,

$$E(X_{i,j}) = \Pr(X_{i,j} = 1) = 2/(j - i + 1).$$

Analysis of Randomized Quicksort

- What is the probability that z_i was compared to z_j?
- What is the probability that z_i or z_j is the first chosen pivot from $Z_{i,j}$?
- Since $|Z_{i,j}| = j - i + 1$,

$$E(X_{i,j}) = \Pr(X_{i,j} = 1) = 2/(j - i + 1).$$

Analysis of Randomized Quicksort

$$
\begin{aligned}
E(X) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(X_{i,j}) \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2/(j-i+1) \\
&= 2 \cdot \sum_{i=1}^{n-1} \left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-i+1} \right) \\
&\leq 2 \cdot \sum_{i=1}^{n-1} \left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \right) = 2(n-1)H_n.
\end{aligned}
$$

Analysis of Randomized Quicksort

$$\begin{aligned} H_n &= \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \\ &\leq \int_1^n \frac{1}{y} dy \\ &= \ln n - \ln 1 = \ln n \end{aligned}$$

- $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ is the harmonic series.
- H_n is $\Theta(\log n)$.

$$E(X) = 2(n-1)H_n = \Theta(n \log n).$$

Analysis of Randomized Quicksort

$$
\begin{aligned}
H_n &= \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \\
&\leq \int_1^n \frac{1}{y} dy \\
&= \ln n - \ln 1 = \ln n
\end{aligned}
$$

- $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ is the harmonic series.
- H_n is $\Theta(\log n)$.

$$E(X) = 2(n-1)H_n = \Theta(n \log n).$$

Analysis of Randomized Quicksort

$$
\begin{aligned}
H_n &= \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \\
&\leq \int_1^n \frac{1}{y} dy \\
&= \ln n - \ln 1 = \ln n
\end{aligned}
$$

- $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ is the harmonic series.
- H_n is $\Theta(\log n)$.

$$E(X) = 2(n-1)H_n = \Theta(n \log n).$$

Randomized Quicksort

Theorem

Randomized Quicksort correctly sorts the input array in-place and requires $\Theta(n \log n)$ comparisons in expectation.

- Can still take $\Theta(n^2)$ time in worst case.
- But with low probability.

- Randomized quicksort is a Las Vegas algorithm.

Randomized Quicksort

Theorem

Randomized Quicksort correctly sorts the input array in-place and requires $\Theta(n \log n)$ comparisons in expectation.

- Can still take $\Theta(n^2)$ time in worst case.
- But with low probability.

- Randomized quicksort is a Las Vegas algorithm.

Randomized Quicksort

Theorem

Randomized Quicksort correctly sorts the input array in-place and requires $\Theta(n \log n)$ comparisons in expectation.

- Can still take $\Theta(n^2)$ time in worst case.
- But with low probability.

- Randomized quicksort is a Las Vegas algorithm.

Randomized Quicksort

Theorem

Randomized Quicksort correctly sorts the input array in-place and requires $\Theta(n \log n)$ comparisons in expectation.

- Can still take $\Theta(n^2)$ time in worst case.
- But with low probability.

- Randomized quicksort is a Las Vegas algorithm.

Outline

1. Introduction
2. Polynomial Identity Testing
3. Randomized Quicksort
4. Randomized Min-Cut
5. Finally

Global Min-Cut

A cut of a graph is a set of edges, which when removed, disconnects the graph. Given a connected undirected graph $G = (V, E)$, find a cut which has minimum cardinality.

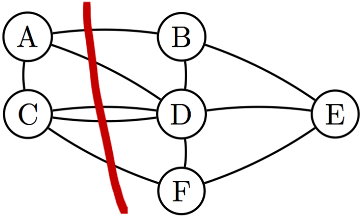


Figure: Courtesy: Andreas Klappenecker

- Applications: Clustering, Network Reliability etc.
- Various deterministic algorithms known
- All are complex to describe

Global Min-Cut

A cut of a graph is a set of edges, which when removed, disconnects the graph. Given a connected undirected graph $G = (V, E)$, find a cut which has minimum cardinality.

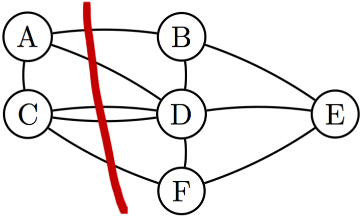


Figure: Courtesy: Andreas Klappenecker

- Applications: Clustering, Network Reliability etc.
- Various deterministic algorithms known
- All are complex to describe

Edge Contraction

To contract an edge $e = \{x, y\}$ of G, we merge the vertices x and y to create a single vertex xy. We retain the multiple edges that may result but don't retain the self loops.

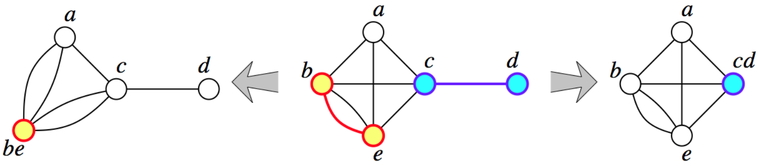


Figure: Courtesy: Jeff Erickson

- The collapsed graph is denoted by G/e.
- G/e need not be a simple graph.
- Contraction can be done in $\Theta(n)$ time.

Edge Contraction

To contract an edge $e = \{x, y\}$ of G, we merge the vertices x and y to create a single vertex xy. We retain the multiple edges that may result but don't retain the self loops.

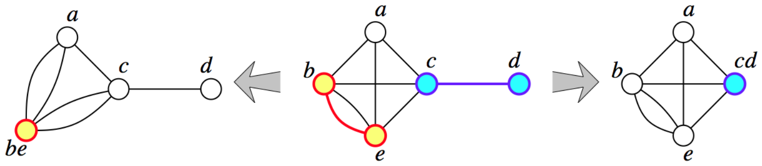


Figure: Courtesy: Jeff Erickson

- The collapsed graph is denoted by G/e.
- G/e need not be a simple graph.
- Contraction can be done in $\Theta(n)$ time.

Edge Contraction

To contract an edge $e = \{x, y\}$ of G, we merge the vertices x and y to create a single vertex xy. We retain the multiple edges that may result but don't retain the self loops.

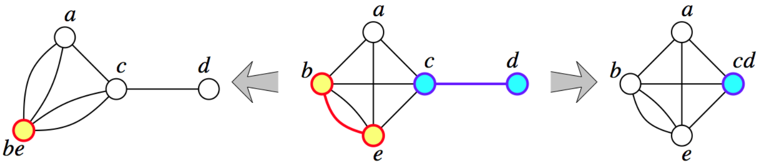


Figure: Courtesy: Jeff Erickson

- The collapsed graph is denoted by G/e.
- G/e need not be a simple graph.
- Contraction can be done in $\Theta(n)$ time.

Karger's Min-Cut Algorithm

Randomized Min-Cut

- Pick an edge $e = \{x, y\}$ at random.
- Contract the edge e and get $G' = G/e$.
- If there are more than 2 vertices, repeat.
- Else, output the edges remaining as your cut.

- Caution: Picking e at random is not the same as picking two connected vertices x, y at random.
- This algorithms completes in $\Theta(n^2)$ time.

Illustration of the Algorithm: Successful

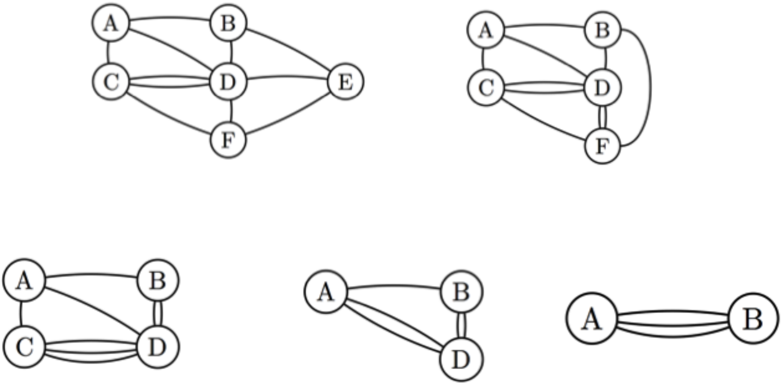


Figure: Courtesy: Andreas Klappenecker

Illustration of the Algorithm: Unsuccessful

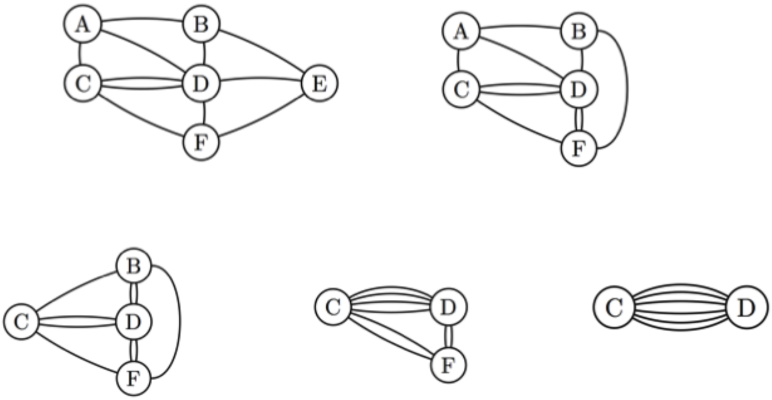


Figure: Courtesy: Andreas Klappenecker

Observations

- A cut of G' is a cut of G.
- The min-cut size of the successive graphs never decrease.
- The algorithm returns a cut of the graph.
- The cut need not be minimal.

Claim 1

Cut C is returned as long as none of the edges $e \in C$ are randomly chosen.

Observations

- A cut of G' is a cut of G.
- The min-cut size of the successive graphs never decrease.
- The algorithm returns a cut of the graph.
- The cut need not be minimal.

Claim 1

Cut C is returned as long as none of the edges $e \in C$ are randomly chosen.

Observations

- A cut of G' is a cut of G.
- The min-cut size of the successive graphs never decrease.
- The algorithm returns a cut of the graph.
- The cut need not be minimal.

Claim 1

Cut C is returned as long as none of the edges $e \in C$ are randomly chosen.

More observations

Claim 2

Let C be a min-cut of G. The probability that an edge in C is contracted in the first step is at most $2/n$.

Proof

- If $|C| = k$, then all vertices have degree at least k.
- Else the single vertex can form a cut smaller than C.
- Total number of edges $|E| \geq kn/2$.
- Pr(edge $e \in C$ is chosen) $\leq \frac{k}{kn/2} = 2/n$.

More observations

Claim 2

Let C be a min-cut of G. The probability that an edge in C is contracted in the first step is at most $2/n$.

Proof

- If $|C| = k$, then all vertices have degree at least k.
- Else the single vertex can form a cut smaller than C.
- Total number of edges $|E| \geq kn/2$.
- Pr(edge $e \in C$ is chosen) $\leq \frac{k}{kn/2} = 2/n$.

More observations

Claim 2

Let C be a min-cut of G. The probability that an edge in C is contracted in the first step is at most $2/n$.

Proof

- If $|C| = k$, then all vertices have degree at least k.
- Else the single vertex can form a cut smaller than C.

- Total number of edges $|E| \geq kn/2$.

- $\Pr(\text{edge } e \in C \text{ is chosen}) \leq \frac{k}{kn/2} = 2/n$.

More observations

Claim 2

Let C be a min-cut of G. The probability that an edge in C is contracted in the first step is at most $2/n$.

Proof

- If $|C| = k$, then all vertices have degree at least k.
- Else the single vertex can form a cut smaller than C.
- Total number of edges $|E| \geq kn/2$.
- Pr(edge $e \in C$ is chosen) $\leq \frac{k}{kn/2} = 2/n$.

More observations

Claim 3

Let C be a min-cut of G. If C remains a cut of the graph after i steps, the probability that an edge in C is contracted in step $i+1$ is at most $2/n - i$.

Proof

- If C remains a cut after i steps, then it is a min-cut of the graph.
- Since C is a min-cut, total number of edges $|E| \geq |C|(n-i)/2$.
- $\Pr(\text{edge } e \in C \text{ is chosen}) \leq \frac{|C|}{|C|(n-i)/2} = 2/(n-i)$.

More observations

Claim 3

Let C be a min-cut of G. If C remains a cut of the graph after i steps, the probability that an edge in C is contracted in step $i+1$ is at most $2/n-i$.

Proof

- If C remains a cut after i steps, then it is a min-cut of the graph.
- Since C is a min-cut, total number of edges $|E| \geq |C|(n-i)/2$.
- $\Pr(\text{edge } e \in C \text{ is chosen}) \leq \frac{|C|}{|C|(n-i)/2} = 2/(n-i)$.

More observations

Claim 3

Let C be a min-cut of G. If C remains a cut of the graph after i steps, the probability that an edge in C is contracted in step $i+1$ is at most $2/n-i$.

Proof

- If C remains a cut after i steps, then it is a min-cut of the graph.
- Since C is a min-cut, total number of edges $|E| \geq |C|(n-i)/2$.
- $\Pr(\text{edge } e \in C \text{ is chosen}) \leq \frac{|C|}{|C|(n-i)/2} = 2/(n-i)$.

More observations

Claim 3

Let C be a min-cut of G. If C remains a cut of the graph after i steps, the probability that an edge in C is contracted in step $i+1$ is at most $2/n-i$.

Proof

- If C remains a cut after i steps, then it is a min-cut of the graph.
- Since C is a min-cut, total number of edges $|E| \geq |C|(n-i)/2$.
- $\Pr(\text{edge } e \in C \text{ is chosen}) \leq \frac{|C|}{|C|(n-i)/2} = 2/(n-i)$.

Success Probability

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

Proof

- C remains in the graph if none of its edges are chosen till step $n-2$.
- Probability that none of its edges are chosen in any step is

$$\begin{aligned} &\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\cdots\left(1-\frac{2}{3}\right) \\ &= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{1}{3}\right) = \frac{2}{n(n-1)} \end{aligned}$$

Success Probability

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

Proof

- C remains in the graph if none of its edges are chosen till step $n-2$.
- Probability that none of its edges are chosen in any step is

$$
\begin{aligned}
&\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\cdots\left(1-\frac{2}{3}\right)\\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{1}{3}\right) = \frac{2}{n(n-1)}
\end{aligned}
$$

Success Probability

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

Proof

- C remains in the graph if none of its edges are chosen till step $n-2$.
- Probability that none of its edges are chosen in any step is

$$\begin{aligned} &\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\dots\left(1-\frac{2}{3}\right) \\ &= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\dots\left(\frac{1}{3}\right) = \frac{2}{n(n-1)} \end{aligned}$$

Boosting

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

- We can improve this by repeating the algorithm

$$\Pr(C \text{ is not chosen in any of } t \text{ trials}) \leq \left(1 - \frac{2}{n(n-1)}\right)^t$$

- Setting $t = n(n-1)/2$ gives us that the probability of failure is $\leq 1/e$.
- We can boost even further using more repeats.

Theorem

If C is a min-cut of G, then the probability that any of the $n(n-1)/2$ repeated trials of the algorithm does not output C is at most $1/e$.

Boosting

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

- We can improve this by repeating the algorithm

$$\Pr(C \text{ is not chosen in any of } t \text{ trials}) \leq \left(1 - \frac{2}{n(n-1)}\right)^t$$

- Setting $t = n(n-1)/2$ gives us that the probability of failure is $\leq 1/e$.
- We can boost even further using more repeats.

Theorem

If C is a min-cut of G, then the probability that any of the $n(n-1)/2$ repeated trials of the algorithm does not output C is at most $1/e$.

Boosting

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

- We can improve this by repeating the algorithm

$$\Pr(C \text{ is not chosen in any of } t \text{ trials}) \leq \left(1 - \frac{2}{n(n-1)}\right)^t$$

- Setting $t = n(n-1)/2$ gives us that the probability of failure is $\leq 1/e$.
- We can boost even further using more repeats.

Theorem

If C is a min-cut of G, then the probability that any of the $n(n-1)/2$ repeated trials of the algorithm does not output C is at most $1/e$.

Boosting

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

- We can improve this by repeating the algorithm

$$\Pr(C \text{ is not chosen in any of } t \text{ trials}) \leq \left(1 - \frac{2}{n(n-1)}\right)^t$$

- Setting $t = n(n-1)/2$ gives us that the probability of failure is $\leq 1/e$.
- We can boost even further using more repeats.

Theorem

If C is a min-cut of G, then the probability that any of the $n(n-1)/2$ repeated trials of the algorithm does not output C is at most $1/e$.

Boosting

Theorem

If C is a min-cut of G, then the probability that any of the $n(n-1)/2$ repeated trials of the algorithm does not output C is at most $1/e$.

- In $\Theta(n^4)$ time we can get the probability of failure to any constant by further repeats.

Counting Min-Cuts

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

- If G has k min-cuts, $C_1, C_2, \ldots, C_k$, then the above theorem can be applied for each C_i.
- Each of these events are mutually exclusive, since the algorithm outputs only one cut.
- The probability of outputting any min-cut is at least $2k/n(n-1)$.
- Since a probability cannot exceed 1, we can conclude that $k \leq n(n-1)/2$.

Counting Min-Cuts

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

- If G has k min-cuts, $C_1, C_2, \ldots, C_k$, then the above theorem can be applied for each C_i.
- Each of these events are mutually exclusive, since the algorithm outputs only one cut.
- The probability of outputting any min-cut is at least $2k/n(n-1)$.
- Since a probability cannot exceed 1, we can conclude that $k \leq n(n-1)/2$.

Counting Min-Cuts

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

- If G has k min-cuts, $C_1, C_2, \ldots, C_k$, then the above theorem can be applied for each C_i.
- Each of these events are mutually exclusive, since the algorithm outputs only one cut.
- The probability of outputting any min-cut is at least $2k/n(n-1)$.
- Since a probability cannot exceed 1, we can conclude that $k \leq n(n-1)/2$.

Counting Min-Cuts

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

- If G has k min-cuts, $C_1, C_2, \ldots, C_k$, then the above theorem can be applied for each C_i.
- Each of these events are mutually exclusive, since the algorithm outputs only one cut.
- The probability of outputting any min-cut is at least $2k/n(n-1)$.
- Since a probability cannot exceed 1, we can conclude that $k \leq n(n-1)/2$.

Counting Min-Cuts

Main Theorem

If C is a min-cut of G, then the algorithm outputs C with probability at least $2/n(n-1)$.

- If G has k min-cuts, $C_1, C_2, \ldots, C_k$, then the above theorem can be applied for each C_i.
- Each of these events are mutually exclusive, since the algorithm outputs only one cut.
- The probability of outputting any min-cut is at least $2k/n(n-1)$.

- Since a probability cannot exceed 1, we can conclude that $k \leq n(n-1)/2$.

Outline

Some points to take-away

- Randomized algorithms usually lead to extremely simple algorithms to describe and program.
- The power of randomness is not clear, it is not clear whether randomized algorithms help us in solving more problems in polynomial time.
  BPP, RP and ZPP are a few randomized polynomial time complexity classes. We do not know if any one of them is distinct from P.
- However, it seems that randomness makes it easier to solve problems.
- Randomness must be treated as a resource. Pure, unbiased random bits are very expensive to obtain.

Some points to take-away

- Randomized algorithms usually lead to extremely simple algorithms to describe and program.
- The power of randomness is not clear, it is not clear whether randomized algorithms help us in solving more problems in polynomial time.
  BPP, RP and ZPP are a few randomized polynomial time complexity classes. We do not know if any one of them is distinct from P.
- However, it seems that randomness makes it easier to solve problems.
- Randomness must be treated as a resource. Pure, unbiased random bits are very expensive to obtain.

Some points to take-away

- Randomized algorithms usually lead to extremely simple algorithms to describe and program.
- The power of randomness is not clear, it is not clear whether randomized algorithms help us in solving more problems in polynomial time.
  BPP, RP and ZPP are a few randomized polynomial time complexity classes. We do not know if any one of them is distinct from P.
- However, it seems that randomness makes it easier to solve problems.
- Randomness must be treated as a resource. Pure, unbiased random bits are very expensive to obtain.

Some points to take-away

- Randomized algorithms usually lead to extremely simple algorithms to describe and program.
- The power of randomness is not clear, it is not clear whether randomized algorithms help us in solving more problems in polynomial time.
  BPP, RP and ZPP are a few randomized polynomial time complexity classes. We do not know if any one of them is distinct from P.
- However, it seems that randomness makes it easier to solve problems.
- Randomness must be treated as a resource. Pure, unbiased random bits are very expensive to obtain.

References

- Michael Mitzenmacher and Eli Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, Cambridge University Press, 2005.
- Rajeev Motwani and Prabhakar Raghavan, *Randomized Algorithms*, Cambridge University Press, 1997.
- Various lecture notes.

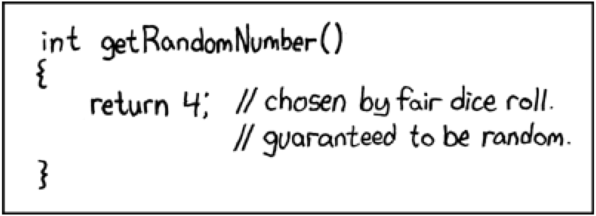


Figure: XKCD Webcomic by Randall Munroe

Thank You